The IULS Approach to Software Wrapper Technology for Upgrading Legacy Systems

Dr. David Corman
The Boeing Company

This article describes using software wrappers in Incremental Upgrade of Legacy Systems as a key technology for modernizing legacy systems. It introduces three types of wrappers, describes a process for selecting which upgrade path to utilize, and discusses a tool-set developed by The Boeing Company for the Air Force that automatically generates wrappers. Lastly, it discusses real-world avionics upgrade examples where the tool-set has been applied and its effectiveness.

Avionics upgrades are frequent and occur for many reasons, including warfighting enhancements, countering changing threats, hardware obsolescence, and computer resource under-capacity. A typical production avionics upgrade cycle for military aircraft frequently involves embedded software changes. New versions of mission processor software, the most volatile class of avionics software, are typically released annually and take two years to field from initial definition.

Hardware obsolescence occurs collectively over a longer term as vendors change their business (military/commercial mix) and technology. Software tools and technology also evolve over a longer period but may be driven by short-term events such as the introduction and imposition of Ada. The change cycles are not synchronized so the optimal hardware, software and tool technology, and respective program funding to support an avionics upgrade at a given point in time are often not available.

The problem of cost-effectively upgrading legacy systems can be mitigated through reengineering with the latest generation hardware and architectural concepts, including object-oriented (OO) software design, which inherently contains and isolates change. However, legacy avionics software represents a large investment in development tools, executable code, and ground and flight qualification. Should the upgrade require complete reengineering of this legacy software, much of this investment is lost; many aircraft programs simply cannot afford the up-front costs associated with reengineering and complete re-qualification.

One solution to this dilemma is implementing reengineering incrementally by inserting the latest technology in smaller, affordable steps, thereby reducing risk and deferring or reducing cost. Software wrapper technologies hold particular promise in meeting this challenge.

A wrapper is a software adapter or shell that isolates a software component from other components and its processing environment (its context). The wrapped component becomes a software object. Its operational capability (functions and data) is encapsulated, and it can be integrated through its standard interface with other software objects to form an operational flight program (OFP) on a single or distributed processor host. The wrapper manages the timeliness of all shared and external data, and provides any necessary transformations.

For upgrades, the goal is to develop the new or reengineered applications using the latest software engineering techniques such as OO design and languages (Ada and C++) with minimal concessions to the internal structure of the legacy system. It is developed as if all other applications were resident in the new environment. Because the new software is written within the paradigms of OO design and languages, the wrapper can eventually be removed once all of the application functions have migrated to the new system. At this time, the legacy system can also be removed.

The Incremental Upgrade of Legacy Systems (IULS) program is a research and development effort, whose main objective is to develop, demonstrate, and transition software wrapper technology that will enable cost effective, incremental improvements to fielded weapon system avionics. The products of IULS are: 1) methodology for analyzing software upgrade approach, 2) wrapper technology, 3) tool-set for constructing wrappers for software upgrades, and 4) demonstrations of IULS wrapper technology applied to three significantly challenging problems: F-15E, C-17, and CV-22 avionics.

IULS is funded by the Air Force Research Laboratory, Embedded Information Systems Engineering Branch (AFRL)/(IFTA). Participants in the project include The Boeing Company, Honeywell Technology Center, General Dynamics Information Systems, and TRW-Dayton.

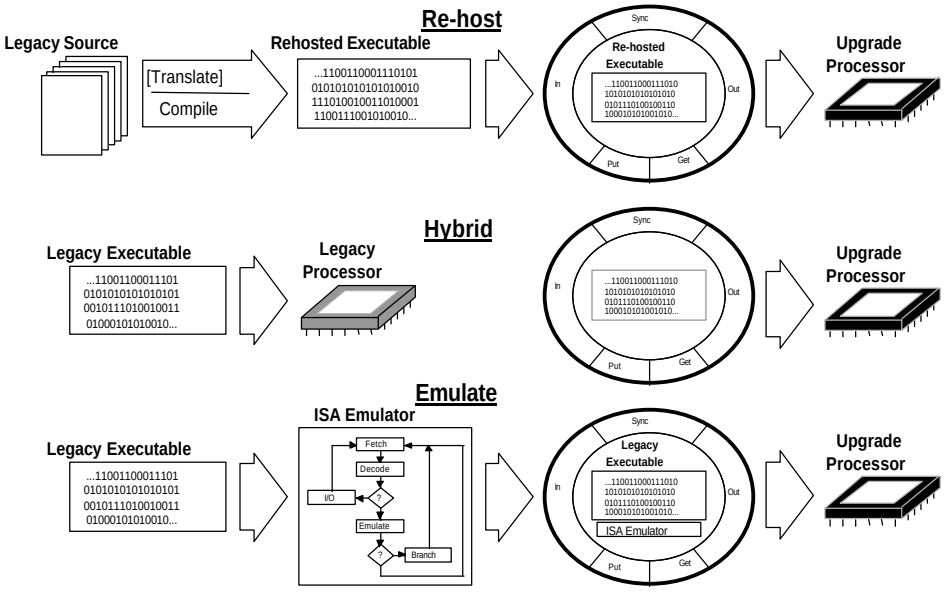
Software Wrapper Technologies

Figure 1 illustrates three hypothetical cases of implementing software changes using wrappers.

Re-Host

In the *re-host* case, the legacy processor is obsolete and/or its resources are insuffi-

Figure 1: *Wrapper Cases*



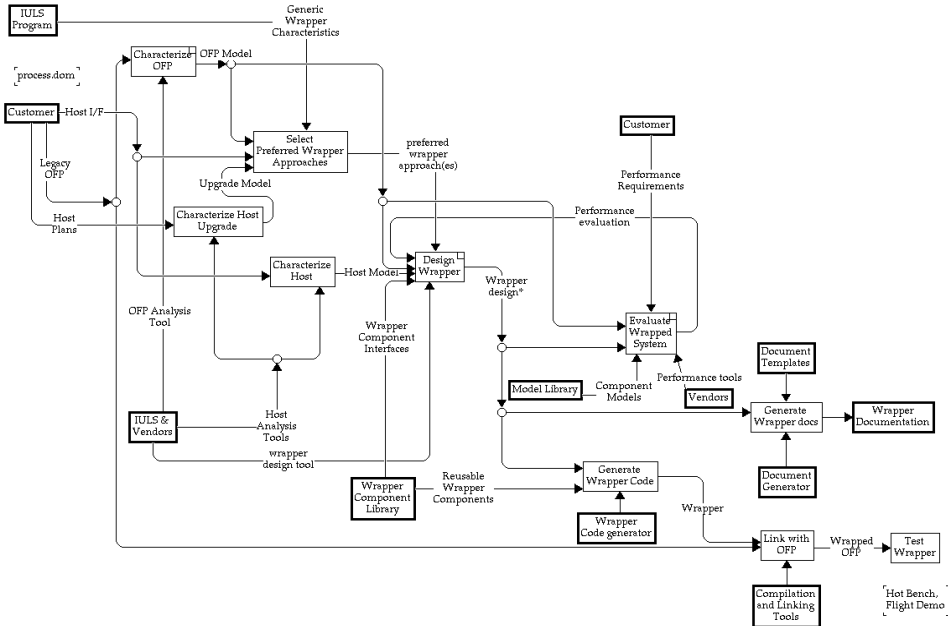


Figure 2: *Nominal Legacy Operational Flight Program Wrapper Process*

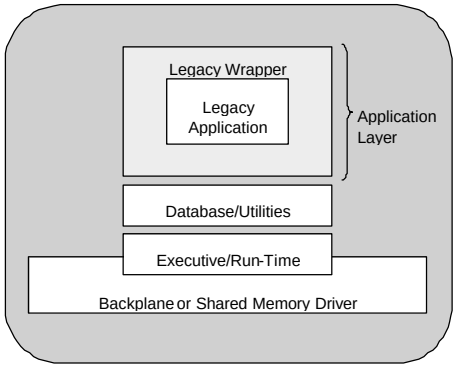
cient to support additional upgrades. The legacy software is re-hosted to a new processor by translating its source code (e.g., Ada 83 to C++) and/or recompiling it for the new target (e.g., Ada 83 to Ada 95). Reengineering the OFP on the new processor could not be justified so wrapper components are added to make it *look like* an object in the OFP. New software features can be added incrementally to the wrapped component, or preferably, designed as new objects in the OFP.

Hybrid

In the *hybrid* case, the legacy processor and its OFP are retained for various reasons (high reengineering or logistics costs, etc.), but its resources are insufficient to support additional upgrades. Also, there is an opportunity to satisfy upgrade requirements with reuse library components that are developed with more modern languages (such as Ada 95 or C++) and tools.

New features can be added incrementally to the upgrade OFP as objects on a new processor. The objects will be bridged to the legacy OFP and processor with wrapper components. As components in the legacy OFP require changes, they can be reengineered and moved to the new processor. At some point in the migration, the remaining legacy components are re-hosted, the legacy processor is upgraded or discarded, and the wrapper components in the new OFP, associated with the legacy OFP interfaces, can be removed.

Figure 3: *Wrapper Software Structure*



Emulate

Obsolete or underpowered hardware is also addressed in the *emulate* case. The legacy software is judged to be very costly to reengineer and/or re-qualify. The object code is executed on the new processor by an emulation of the legacy processor's instruction set architecture.

Changes can still be made to the legacy executable using the legacy compiler and software engineering environment. The emulator and other wrapper components make the legacy executable component (binary) *look like* an object. Other feature upgrades could be added as new objects on the new processor.

Wrapping Process

As with any other software development activity, wrapper creation follows a process, shown in the IDEF0 diagram in Figure 2, and is automated with tools. In an IDEF0 diagram, consumed inputs (e.g., data files) go in the left side of an activity box; generated outputs (e.g., completed design objects) emerge from the right side; constraints (e.g., requirements, schedules) go in the top; and mechanisms (e.g., tool support) go in the bottom. The following subsections describe tool mechanisms that support the wrapper design process and the data that flows between them.

Wrapper Implementation Considerations

Wrappers are generally applied at the application domain level. They act as clients and servers to the encapsulated component. Figure 3 illustrates a general wrapper structure for an OFP on a single processor. The legacy application interfaces with other applications and other layers only through the wrapper. The wrapper architecture is tailored to the specific legacy OFP environment.

The method selected to implement the upgrade of a legacy system is to an extent an economic decision. The emulation option will tend to favor a context of a stable application, infrequent OFP modification, and obsolete hardware. These cases will generally be lower in performance, being older systems. Therefore, the context analysis would focus on issues of throughput, OFP stability, parts obsolescence, OFP utility, etc.

Selecting a hybrid or re-host approach would generally be appropriate for a more dynamic and/or higher performance system. Typical context would be OFPs that are subject to periodic update and version release. In addition, the higher the throughput needed, the more likely the upgrade path will not include emulation. The reason for this analysis factor is that while *software emulation* provides a growth path for the legacy upgrade, there is a penalty paid for using processing resource overhead. A hardware emulator approach will, with time, become a technology dead-end (as would be the case for the legacy system) and require more near term upgrade effort.

Any selected legacy upgrade technique will, of course, be subject to obsolescence and eventual upgrading. Therefore, the methods described for the upgrade process and wrapper generation emphasize as flexible an approach as possible. Also, the methods need to be portable between hardware platforms since these components will change rapidly (on the order of months vs. years). Therefore, the initial system context analysis would focus on OFP issues such as throughput, stability, etc.

Using the IULS Tool-Set in Upgrading

As part of the IULS program, Boeing and its IULS teammates developed a set of guidelines, processes, and tools to help the avionics engineer determine and implement a *best* wrapper upgrade strategy. The

IULS tool-set that was developed in this effort has played a major role in automating the upgrade approach in the re-host/hybrid domains. The tool-set is used to iteratively develop high-level and detailed models of the OFP model, the host model, and the upgrade model. Figure 4 displays some of the basic elements of the IULS graphical tool-set.

Briefly, the tool provides a graphical capability to model the legacy and upgraded system, including data interfaces and constraints. It includes a re-use component library that can be used to meet requirements common to avionics applications. It provides code and documentation generation capability to construct and document the software wrapper used to bridge the legacy and upgraded system. Also, the tool provides system-modeling capabilities that can be applied to validate system performance against scheduling constraints common to hard real-time avionics systems.

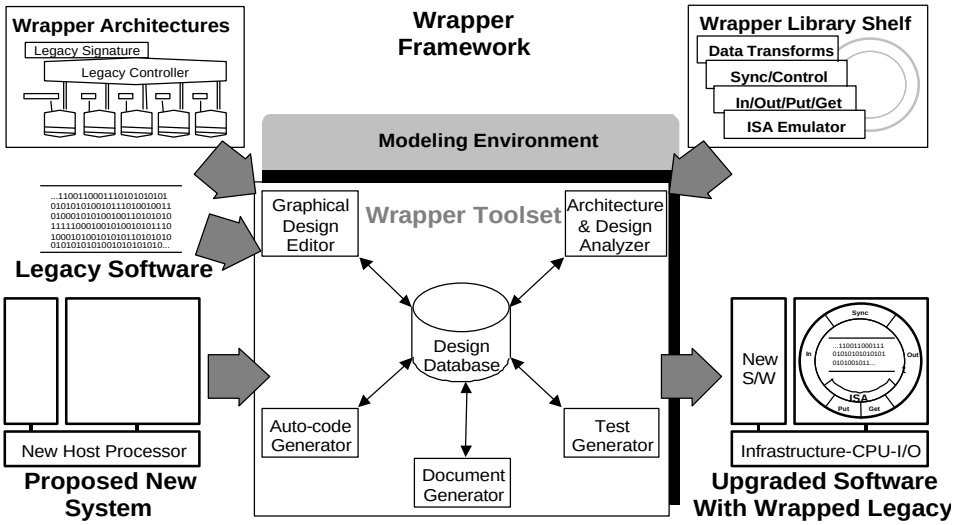


Figure 4: *Graphical Tool-Set*

Customer Inputs to the Process

The *select preferred wrapper approaches activity* consists of selecting from among the three basic wrapper approaches. The host plans, legacy OFP, and host interface (I/F) are information supplied by the customer. The information specifies the OFP re-hosting problem in a manner that is sufficient to trigger the next activities. The host plans identify the need to re-host the OFP onto a new target platform and are used to select one of the three wrapper approaches, or possibly narrow the selection down to two of the three approaches that will be considered during the wrapper design process.

Characterizing the Problem

The IULS tool-set is used to characterize and model the legacy system OFP. The resulting *OFP model* is a description of the legacy code, it's source language, the available documentation and compilers, and the description of the I/O required by the OFP, including timing. The model describes each interface modality that the OFP has with other OFPs and with the legacy host hardware. The tool-set is also used to develop the *host model*, which provides a description of the target host computer environment (not the legacy host). It includes a description of the machine code, compilers available, event capabilities, I/O capabilities, and kernel operating system (OS) interfaces.

The *upgrade model* is a description of the plans to upgrade the target host computer in the future. The plans for future upgrades influence the choice of wrapper approaches.

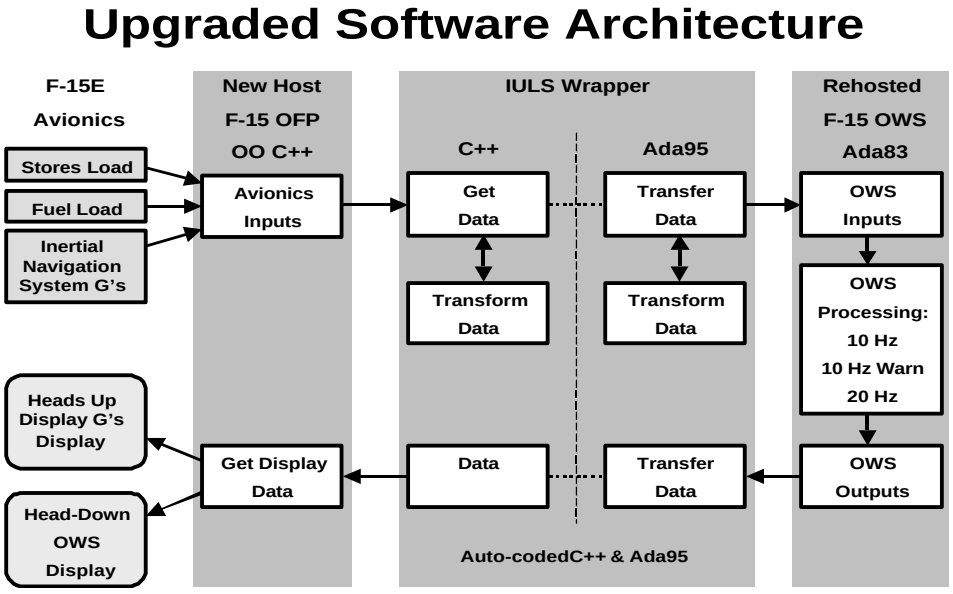
The characterization step provides high-level information that can be used to perform coarse trade-off analyses that contribute toward a final selection of wrapper approach as well as some of the basic decisions about the wrapper design. Some of this high-level model will identify wrapper components able to be tailored from a reuse library.

Wrapper Design

As shown in Figure 4, the wrapper is designed using the OFP model, host model, and upgrade model, as well as the *reusable wrapper components*. The IULS tool-set includes a set of reusable wrapper components that are placed *on the shelf* and can be either quite general in nature (e.g., format converters from one process to another) or domain specific (data access methods for an aircraft). The reusable wrapper components are linked to requirements and test cases that are also reused. The process of wrapper design includes specifying the following within the tool-set:

- Invocation interface: This is the interface for entering into and returning from the OFP code. It includes the description of error handling interfaces.
- Semaphores and interrupt handlers: This is the interface that defines synchronous process behavior, both hardware supported and software only.
- Data accessors: This interface defines accessors for data objects that the OFP shares with other software modules.
- Emulator configuration: This describes the emulator that may be interfaced to the wrapper, depending on the wrapper approach used.
- Object adapter: This is software that makes the wrapped OFP look like one

Figure 5: *OFP Wrapper for F-15 Demonstration*



Component	Software Lines of Codes (Not Comment/Blank)	Total Source Lines
Total OFP (C++ and Ada)	119,363	534,054
OWS Application (Ada Including PIMs)	7,195	23,738
Ada Wrapper	482	880
C++ Wrapper	408	811

Table 1: *Wrapper Component Sizes*

or more objects to an underlying object request broker (ORB). It defines the OO interface classes and methods comprised in the wrapper.

- Legacy port design: If the wrapper approach calls for re-hosting the legacy OFP, then a description of the tools, process, and wrapper interface objects needed to support that approach are specified.

Evaluation of the Wrapped OFP

After a candidate wrapper is designed and a baseline determined, it is evaluated. The evaluation is performed using *component models* selected from a library. The component models are the representations of the target hardware system. These are used in running simulations of the target system to determine performability. System modeling and evaluation tools such as Cosmos and Foresight can be used in concert with the IULS tool-set to simulate and evaluate the wrapper design.

Wrapper Code Generation

Once the wrapper design has successfully passed evaluation tests, the IULS tool-set provides the capability to automatically generate the *wrapper* using a code generator. It identifies the versions of the components, how those components were tailored, and how those components are to link together with the OFP. The wrapper is then linked with the OFP to create the *wrapped OFP*.

OFP Integration, Test, and Documentation

The wrapped components and host components are compiled, linked, and integrated into the target processor system. System and software testing is performed to a level appropriate to the avionics application. By using the same specification for wrapper design, evaluation, and implementation, traceability is greatly simplified and facilitated. Model objects can be cross-referenced to requirements, implementation components, evaluation models, tests, and test results. The IULS tool-set includes a document generator that compiles software documentation from the design database using customizable templates.

Demonstrations

The IULS program included major demonstrations of two of the wrapper approaches: modified re-host and emulation. The demonstrations provided an opportunity to *test and tune* the tool-set in a real-world avionics upgrade environment. The following subsections describe the results of the application of the IULS tool-set.

F–15 Demonstration

For the F-15 demonstration, the IULS problem was to port a legacy F-15 Ada 83 OFP component, the overload warning system (OWS), into a F-15 OFP written in C++ running on a new commercial off-the-shelf (COTS) PowerPC processor. A wrapper was designed and auto-generated using the IULS tool-set. This was the first application of the IULS tool-set and provided us a framework for testing and tuning of the tool.

Figure 5 (see page 11) shows elements of the wrapper design. In particular it shows how the wrapper supports transfer of data from the OO C++ domain into process interface messages understood by the legacy Ada software. The wrapper also performs necessary data transforms and includes a bridge from the Ada 83 software to the C++ and display processes.

The F-15 IULS activity culminated in a successful live flight demonstration conducted on Dec. 1, 1999. The demonstration flight plan called for execution of six test points. These corresponded to combinations of three different weapon loads with two different fuel configurations. The pilot, weapon systems officer, and flight test engineer reported successful test results.

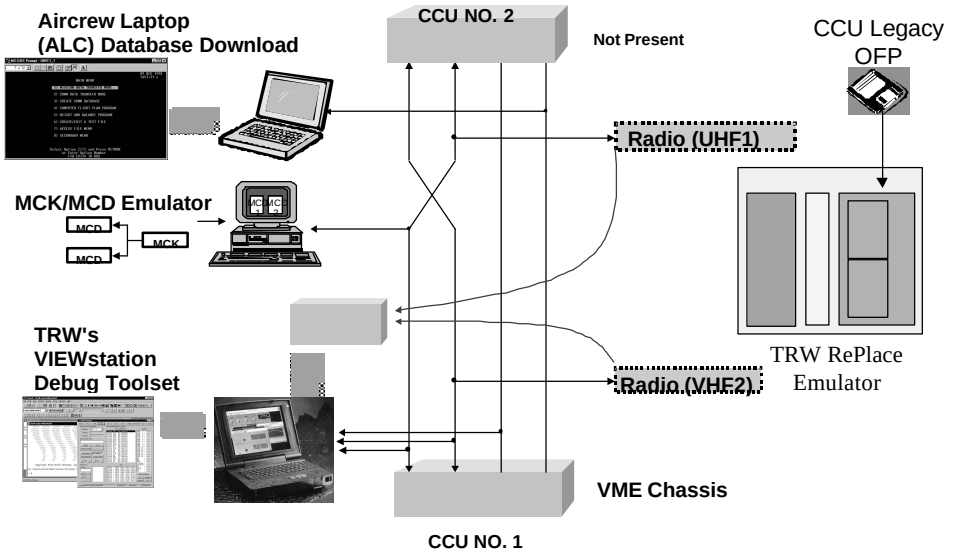
The relative sizes of the components (in source lines of code) for the final demonstration and flight test OFP are shown in Table 1.

We collected metrics on the wrapped system to measure wrapper overhead. It was indicated that wrapped system added an average of 0.36 msec and 0.16 msec to the OWS application execution timelines for the 20 Hz and 10 Hz tasks, respectively, for operation on a PowerPC 603E. The computations were all performed within the time-frame requirements for the OFP.

The F-15 demonstration thoroughly validated the IULS re-host process and tool-set. Operationally, the demonstration received enthusiastic endorsement from the flight crew who referred to it as a *home run* in the post flight debrief. The in-flight performance was 100 percent in agreement with the a priori estimates matching all six test points, exactly. The WrapidH tool proved to be extremely valuable in developing the wrapper design, and the automated code generator worked as expected in both the Ada and C++ domains.

As predicted, considerable domain expertise was required to develop the wrapper. However, IULS engineers who initially had no familiarity with the heritage code performed the bulk of this work. These engineers were able to readily understand the legacy Ada and Common Operational Flight Program (COFP) C++ to the extent required to support wrapper design and system de-bug. Wrapper testing confirmed the prediction that wrapped code integrity would be intact – no problems were detected in which wrapped code operation was an issue. Wrapper overhead was not sub-

Figure 6: *C-17 Demonstration for Emulation Wrapping*



stantial and confirmed system modeling conducted during phase one of the IULS program, which had predicted system throughput was more than adequate for the demonstration requirements.

C–17 Emulation Demonstration

The IULS emulator approach was demonstrated by *wrapping* a legacy C-17 radio control function (RCF) executable (JOVIAL source, MIL-STD-1750A object) with a 1750A ISA emulator running on a PowerPC processor that represented an upgraded communications control unit. The emulator interface and processor context wrappers were generated with TRW's RePLACE tool-set. Figure 6 shows the demonstration concept.

A COTS replacement box (CRB) was constructed, including COTS processor (PowerPC). The emulation engine and RCF OFP were loaded onto the CRB. The CRB operated in concert with the 2nd communications control unit (CCU). This provided a timing challenge since handshaking, normally performed by two 1750A processors across the 1553 interface, was now being performed by the CRB and one CCU. The system was demonstrated in the C-17 avionics laboratory with production test cases and avionics system hardware. The demonstration showed an approximately 90 percent growth capacity for the CRB

The emulation tool-set is being transitioned to the C-17 as part of the on-going communications open systems architecture engineering and manufacturing development program. In this application, emulation is being extended as part of a larger open system upgrade. In particular, a new C++ native language executive is being developed that will then make *calls* to selected (emulated) legacy components. In effect, the emulated legacy software functions as a library of callable functions. This is in contrast to the standard emulation wrapper in which what is simply desired is to execute the legacy software on a more modern processor.

The lesson in transiting the emulation wrapper is that quite frequently programs will apply tools in a different manner than were originally planned, and the tool-set needs to provide inherent flexibility.

CV–22 Demonstration

We are applying the *re-host* wrapper approach with a twist during an on-going demonstration with the CV-22 program. Figure 7 shows the basic elements of the demonstration. The challenges are twofold: First, migrate Ada JASS-JVX Avionics

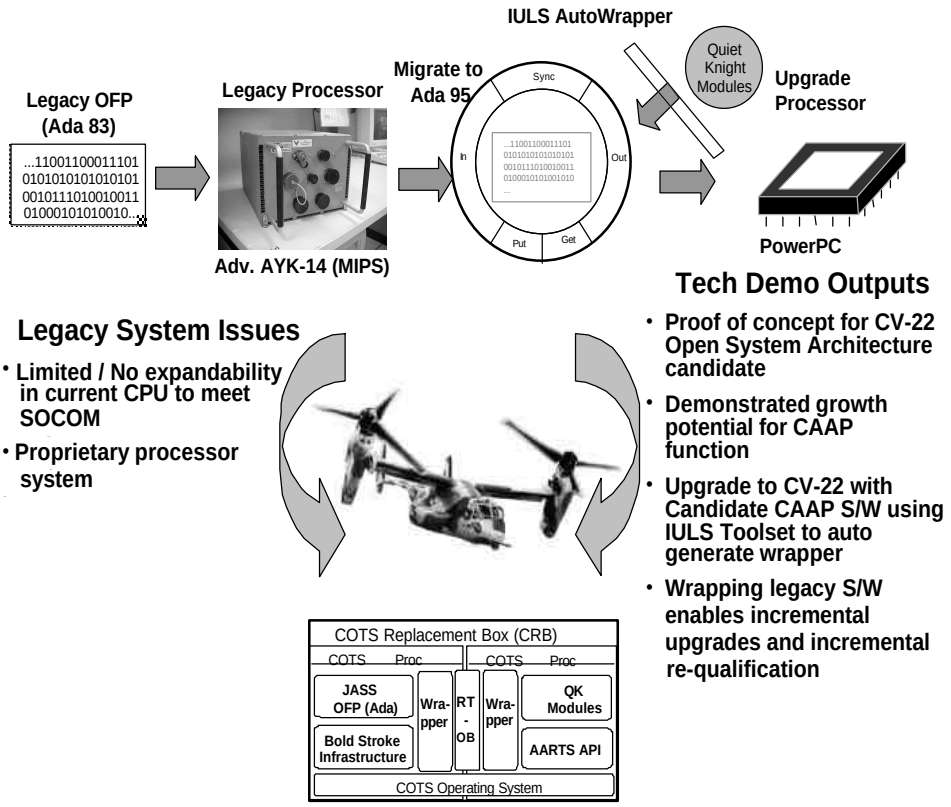


Figure 7: *IULS CV-22 Wrapper Demonstration*

System Software mission software from proprietary processor and system architecture to a COTS PowerPC processor and VME architecture residing under the Boeing Bold Stroke software infrastructure. Second wrap a new application (Quiet Knight – Ada 95) to be executed on a second COTS processor using the tool-set and object request broker (ORB) for distribution.

The CV-22 demonstration is a work in progress. At the time this article was written, the demonstration is planned for December 2001. The wrapper tool-set is being used to develop the ORB interface definition language to construct the wrapper. Initial results have been very promising. The mission OFP has largely (99 percent) been ported to the COTS processor, and the IULS tool-set has been used to generate the wrapper software for the ORB.

Summary

This article has described several important applications of software wrappers to legacy software modernization. Under the AFRL-sponsored IULS program, Boeing developed a tool-set that can be made available to qualified requestors from AFRL/IFTA to support avionics upgrade opportunities. Wrappers are not a panacea for every modernization need. However, experience to date indicates that they can be an important element within the upgrade process.◆

About the Author

David Corman, Ph.D., is a technical fellow of The Boeing Corporation. Dr. Corman is currently working on a variety of projects that focus on upgrading of legacy systems. He was the lead engineer on the Incremental Upgrade of Legacy Systems (IULS) program that resulted in the development of a wrapper tool-set that has been demonstrated in real-time applications including the F-15, C-17, and CV-22. Dr. Corman has more than 20 years experience in software development, including real time, mission planning, C4I systems, and avionics domains, and holds bachelor's and master's degrees in systems science and mathematics, and applied mathematics from Washington University in St. Louis. He earned a doctorate in electrical engineering from the University of Maryland, College Park.

The Boeing Company
P.O. Box 516
St. Louis, MO 63166
Phone: (314) 234-3725
E-mail: david.e.corman@boeing.com